LINGOL - A PROGRESS REPORT

Vaughan R. Pratt
Massachusetts Institute of Technology
Cambridge, MA, U.S.A.

Abstract

A new parsing algorithm is described. It is intended for use with advice-taking (or augmented) phrase structure grammars of the type used by Woods, Simmons. Heidorn and the author. It has the property that it is guaranteed not to propose a phrase unless there exists a continuation of the sentence seen thus far, in which the phrase plays a role in some surface structure of that sentence. The context in which this algorithm constitutes a contribution to current issues in parsing methodology is discussed, and we present a case for reversing the current trend to ever more complex control structures in natural language systems.

L INTRODUCTION

LINGOL (LINGuistics Oriented programming Language) is a language to facilitate the writing of natural language processing programs. In a previous paper [Pratt 19731 the LINGOL system was described, examples of output from several small-scale LINGOL programs were given, and the reader was led through a console session as a trivial-scale French translator was developed. That paper sought to establish LINCOL's credentials as a pedagogical device for newcomers to the art of writing natural language "front ends".

LINGOL was originally conceived as a language intended for use by serious researchers Due to its author's preoccupation with more mathematical pursuits during the past few years, LINGOL has not been exercised until recently with anything but small-scale student-generated programs Since the appearance of [Pratt 1973), several more such small-scale LINGOL programs have been written. More recently, the author has begun work on a large-scale program to see whether LINGOL really can be used as the research tool it was originally designed to be, without compromising those features that made it attractive to beginners (ease of use and low resource consumption) In this paper we concentrate on the implementation of the current version of LINGOL. This discussion will complement the one in [Pr.tt 1973], which concerned the motivation for the LINGOL language, that part of the LINGOL system accessible to the user. Primarily, we shall present a new context-free parsing algorithm that, paradoxically perhaps, is responsible for the efficiency of the very context-dependent parser in LINCOL's cognitive component. The idea behind this algorithm is one that may be of value in other structure-eliciting type problems besides parsing

An issue that still seems to haunt computational linguistics is that of the syntax/semantics dichotomy. Quite clearly, LINGOL takes approximately the same point of view as [Woods 1969], that surface structure is worth eliciting, and that context-free grammars (or their transition-net equivalent) can play a non-trivial role in parsing. In section 24 we discus a relatively new perspective on this issue which puts this style of parsing in a more favorable liight than people have been willing to view it of late [Riesbeck 1975, Marcus 19751

2__THE_CURRENTLIGOI. LMPLXMXN TAT1QK

2-1—G eneral Qverview

We have already discussed elsewhere [Pratt 1973] the rationale for those features of LINGOL available to the user. In this section we shall talk about what goes on behind the scenes The distinction being drawn here is exactly that of the programmer's manual for a language versus the implementation of a particular compiler for that language In the case of LINGOL the author has found, on occasion, people who are unwilling or unable to draw the distinction, the result is a misconception of what the LINGOL user has to put up with in writing his programs, as opposed to what performance he may expect when running his programs. The distinction has in fact to be drawn even more carefully for LINGOL than for conventional programming languages because for the latter, the usuat choice of operations (arrays, lists, block structure, arithmetic and other operations with well-understood implementations) suggests, to within details of little interest to the programmer, the appropriate implementation of the run-time support Since the programs one writes for LINGOL are highly non-deterministic, and are organized as modules (actors, to use a term in vogue) that do not know with whom they will be communicating until LINGOL connects them together during the processing of a sentence, the LINGOL run-time system's task is appreciably more difficult than that of FORTRAN or ALGOL. The burden of control has been shifted from the grammarian to the system, leaving the grammarian free to concentrate on linguistics. So far, no one has proposed an "obviously" good way to tackle this problem for English, and LINGOL users should be prepared to accept radical changes in LINCOL's internal operation (i.e. parsing algorithm) as progress is made in this area (Since the only legitimate effect of such changes is to improve overall resource consumption, not to compromise correctness of the user's program, this is not really a burden on the user.)

For the benefit of non-readers of [Pratt 1973] we describe briefly the overall organization of LINGOL. The LINGOL system is envisaged as a translator from some natural source language to some target language (natural or artificial) of the user's choice, not necessarily a different language from the source language. There are two phases, one that elicits the surface structure of a sentence and one that produces the desired translation(s). The intention is that issues relevant to determining the intended surface structure versus those of translation should be separated out This corresponds to the recommended practice when translating from English to French, say, of understanding each sentence (naturally taking into account previous sentences) before attempting the translation. No restrictions are made on where the LINGOL programmer draws this boundary, or to what extent the information in the two components is duplicated In fact, he can omit the generative component entirely and put everything in the cognitive component, though at some cost in resource consumption At run time, no firm commitment is made by the cognitive component to a particular choice of surface structure of an ambiguous sentence, allowing the generative component to pick and choose when the cognitive component has not had enough information to decide At present LINCOL users are encouraged to try to make their cognitive component intelligent enough to make the right decision, and so far no LINCOL programs have attempted disambiguation in the generative component. One would expect this to change as people attempt more sophisticated programs.

A LINCOL program is a set of rules each having three components: a context-free rule, a cognitive function and a generative function Their respective roles are as follows The CF rule specifies a general English construction, the cognitive component (or "critic") supplies the expertise about that construction and the generative component supplies the information about the target language that may be relevant to this English construction. (Our tacit assumption of English as the source language reflects LINGOL's applications to date)

It is fashionable these days to want to avoid alt reference to context-free grammars beyond warning students of computational linguistics that they are unfit for computer consumption as far as computational linguistics is concerned In LINCOL, as in ATN's [Woods 1969], their role is different from that in, say, the Harvard Predictive Analyzer [Kuno 1965]. Instead of being used to encode all information about English, they form the basis of a pattern-directed non-deterministic programming language. This strategy has several advantages

(i) It allows the programmer to structure his program as a set of relatively self-contained modules, thereby decreasing the number of things he has to keep in his head at once when looking at a particular part of his program.

(ii) It eliminates much of the testing-for-cases control structure the programmer would need in a non-pattern-driven language.

(iii) Flow of control between modules is confined to the surface structure, radically simplifying the controlling and tracing of computations This is in contrast to systems that require the user to supply considerably more information to control the flow of computation [Riesbeck 1975. Marcus 19751 The apparent subtlety of this advantage belies its importance, and we discuss it further in section 2.1

(iv) Instead of having to identify each possible source of ambiguity and think up a way to deal with it, the user writes "critics" of individual situations and lets LINCOL compare the results of the criticisms as applied to competing situations when an ambiguity arises. This reduces the order of magnitude of programming effort in the resolution of ambiguity from possibly order n^2 to order n, where n is the number of situations that may need to be compared. This is in contrast to the notion of "differential diagnosis" presented in [Marcus, 1975]

(v) LINGOL can optimize the user's program much more effectively if it can identify the context-free component by itself If this component were to be incorporated into the cognitive component, a popular practice these days, the system would not be able to do its own optimization as effectively, and the burden would fall back on the user

The reader wanting more information on items (i) to (iii) is referred back to [Pratt 1973] The worked example illustrates each of these advantages The remaining two items are covered in the following sections

2.2_The-Cognilive Component

In this section we describe the LINGOL parser We first present the algorithm on which it is based, and then show how to use this algorithm to assemble the user's modules and set up communication between them.

Before immersing ourselves in the technical details of the algorithm, let us consider the options open to us The goal for the parser is to build the surface structure intended by the sentence's speaker All it has to go on is the top and bottom of this tree, and the rules (grammar) constraining plausible surface structures A decision must be made as to where growth should begin Two extremes are the top-down approach, in which the tree is grown from its root, and the bottom-up, where growth begins from the bottom, that is, from the words of the sentence. Methods with the flavor of either (or both) of these extremes inherit their name(s). These methods have other aliases in the literature Any scheme that claims to be doing "predictive" analysis, that is, that has expectations about what is coming next and uses those expectations as hypotheses to "drive" the program is essentially a top-down method A program that finds substructures (say conceptual dependency structures [Schank 1970]) and uses them to build bigger structures is a bottom-up method. Terms suggested to the author by R. Moore are "hypothesis-driven" for "top-down" and "data-driven" for "bottom-up" These concepts transcend phrase-structure grammars and may be applied to any system responsible for building an hierarchically organized structure

No matter where the construction begins, we do not know how to carry it out in any straightforward way, even if we want nothing more than to satisfy the context-free rules of our grammar. We always run the risk of letting the construction wander down blind alleys For some grammars and some sentences, the top-down method is less likely to run into cul-de-sacs, but the dual case can also arise It Is hardly surprising, given this state of affairs, to hear people wish that they could build structures both top-down and bottom-up in a way that somehow reduced the overhead. One form of this wish is to request a single algorithm that builds no node a bottom-up method would not consider, nor anything a top-down method would not build. An alternative desideratum might be that no node N be built unless that part of the text seen to date is part of som? sentence having a surface structure in which N participates. For a backup-less parser like LINGOL's present one, this is the strongest possible thing one could ask for as far as exploring cul-de-sacs is concerned One might add to the above the requirement that the parser be able to cooperate with other processes such as tuuiines written by the user

The current implementation of LINGOL achieves all of the above goals To be more precise, every node it builds is built by both the Cocke-Kasami-Younger bottom-up algorithm and the ingenious Earley top-down algorithm, the two algorithms cited in [Aho and Ullman 1972] as the canonical methods for parsing general context-free languages That is, the work performed n the intersection of the work done by each of these methods, at least with respect to proposed phrases Moreover, as each phrase is discovered, LINGOL is able to accept advice from other sources (namely the user's cognitive component) and use it to guide the parse

Roughly speaking, the minimization of searching is accomplished by running the Cocke-Kasami-Younger algorithm and as each phrase is discovered asking an "oracle" whether the Earley algorithm would have discovered it The remarkable thing is that this question can be answered in time independent of the length of the input, without having to go and actually run the Earley algorithm to see what it would have done (The time is proportional to the size of the grammar, but in the LINGOL implementation, asking the question involves no more than forming the logical and of N/36 36-bit vectors for a grammar of N non-terminals. Unlike the large grammars Kuno worked with, a large LINGOL grammar should have only from 100 to 200 non-terminals, LINGOL grammars are not expected to have much information encoded in the context-free component. We are at present exploring a dichotomy for non-terminals, known only to LINGOL's internals and not to the grammarian, that would permit having goals for only very few non-terminals, thereby ensuring that N/36 would remain negligible.)

Before discussing the construction of the oracle, let us sketch the version of the Cocke-Kasami-Younger algorithm we shall use We assume that all rules are of the form either A -> B or A ■> B C . where A . B and C are non-terminals, or of the form A -> a where a is a terminal. The presence of A -> B means that this is not really Chomsky normal form, and allows either the user or some preprocessor to turn an arbitrary grammar into this form using only the trick of replacing all but the first item on the right side of a rule having three or more items by a non-terminal which is itself rewritten to be the replaced non-terminals, and so on until all rules have right sides of length I or 2. The most recent version of LINGOL incorporates a preprocessor for this task, so this normal form ts now a feature solely of the implementation, not of the user's language (In [Earley 1968] the notion of "state" is introduced, which elegantly plays the role of these introduced non-terminals In EM ley's notation, the state AB.CDE plays the role of the nonterminal that replaces the CDE Everything we say in terms of our restricted grammars can be rephrased more elegantly in terms of Earley's states The mam advantage of our notation is that the description of the algorithm is less complicated if the reader is not required to think about arbitrarily "long" states) With this form of grammar we can talk about the left and right sons of binary nodes and the "only" sons of unary nodes. For our purposes it will be convenient to refer to only sons as left sons.

In the following version of the algorithm, we use the notation ?x (where x is a variable) to mean "find all x such that". Thus the code

```
        for ?i+?j=5 do print (i,j)
```

will print all pairs of numbers summing to 5. This avoids cluttering up the algorithm with details of searching that the programming reader will have no difficulty filling in.

We shall employ "between-word" notation for positions rather than "at-word" That is, rather than saying that the first word in the sentence is at position I, we will say it lies between positions 0 and I. This avoids any possible ambiguity when referring to the string lying between positions i and j, and also simplifies naming the common boundary of two concatenated strings. It is also the preferred notation in more recent string processing papers.

The Cocke-Kasami-Younger algorithm is:

```
parse: for each word w between
              positions j-1,j (scanning left to right) do
          (for each rule ?A -> w do note(A,j-1,j);
           for each phrase (?B,?i,j) in turn on the queue do
              (for each rule ?A -> B do note(A,i,j);
               for each rule ?A -> ?C B such that (C,?k,i) has
                                              been built do
                    note(A,k,j))).

note(A,i,j): build(A,i,j); enqueue(A,i,j).
```

We assume as given the primitives that build a node (representing a phrase consisting of a non-terminal, a starting position and an ending position - we shall use "phrase" and "node" interchangeably) and that operate on queues (put on and take off elements).

To make this algorithm run in polynomial time ($O(n^3)$ to be exact) "note" is usually written as:

```
note(A,i,j):  if (A,i,j) not yet built then (build(A,i,j);
                                              enqueue(A,i,j)).
```

A more detailed description of this algorithm appears in [Aho and Ullman 1972] We are concerned here with extending the algorithm

The above suffices for context-free recognition. For parsing, nodes must record (in addition to the three items type, start and end) two additional items, namely which rule was invoked when noting that node, and which phrases are its sons. The former allows us to access the cognitive and generative components associated with the rule at a later date, while the latter allows us to recover the surface structure (Having the rule present makes the syntactic category of the node redundant, and in fact LINGOL omits it.)

We now introduce the oracle. Two things are required to construct this oracle a readily accessible representation of the left-most-character relation, and the notion of a goal We first deal with the goals. Associated with each position in the sentence is a set of goats. A goal is a desired non-terminal. If a phrase of the same type as some goal is discovered starting in the position with which the goal is associated, then that phrase can be used as a right son, in conjunction with some phrase discovered earlier, to form a new phrase. For example, if there is a rule A -> B C and a B has been found ending in position i, then a goal C associated with position i will be created. Later, if a C is found starting in position i, it can be used with B to form an A.

Initially there is one goal, to find a Sentence (assuming this is the distinguished non-terminal of the grammar), which is associated with position 0. As the algorithm runs, other subgoals will be generated. While this sounds like a top-down method, no structures are built top-down - they grow bottom-up and the goals are associated with the frontiers of the growing tree (with the exception of the Sentence goal.) The goals appear just when and where you would expect them to; for example, when reading the sentence "Secretary of State Henry Kissinger announced today the cessation of war in the Middle East", after reading the fifth word it will have set up (inter alia) a goal associated with position 5 and wanting a verb phrase. In fact all goals for a given position are created at once, just before reading the word at that position. We defer for a bit the question of where goals come from, and look first at how they are used.

Let R be a binary relation on the non-terminals of the grammar such that ARB if and only if there exists a rule of the form A -> B... . Then R^*, the transitive closure of R, is the left-most-character relation alluded to above. Thus AR^*B can hold just when there exists a derivation $A \overset{*}{\Rightarrow} B\ldots$, or for those who prefer trees (myself included) it should be possible for B to occur on the left edge of some tree with root A that satisfies the productions of the grammar. (See [Griffit' and Petrick 1965] for other applications of this relation.

It is straightforward to incorporate this oracle and the goal machinery into the algorithm. We assume primitives for storing and fetching goals which are pairs (A,i) of non-terminals and positions. To introduce goals, append to the inmost loop of "parse" the code:

```
for each rule ?A -> B ?C
   such that goal (?D,i) exists satisfying DR*A do
      setgoal(C,j)
```

and precede the code for "note" with the test:

```
if goal(?B,i) exists such that BR*A holds then
```

To see that the algorithm as modified does no less than it has to (i.e that it overlooks nothing), suppose we have an initial segment of a sentence of the language of the given grammar. Then in any tree for this sentence, we claim that all phrases in the tree contained within the initial segment will have been proposed by the time we have reached the end of the segment. We also claim that every right son in the tree will have been generated as a goal just before the parser reached the starting position of that phrase (The induction proofs of these claims are messy and probably inappropriate for this paper - the interested reader is encouraged to fill in his own details.) It follows from these two claims that the oracle will always answer in the affirmative when a phrase thai appears in the tree Is proposed to the oracle, because every phrase is the left-most character of either some non-left-son or of the root, and all such possible goals will have been created by the time we propose this phrase. Hence the phrase is correctly built.

To see that no node is built thai could not participate in some surface structure of some completion of the sentence, we must assume that for each non-terminal of the grammar there exists a derivation starting with that non-terminal and ending with a string of all terminals. This follows immediately if we require that every non-terminal appear in at least one surface structure of some sentence of the language, a perfectly reasonable requirement. Suppose that we have just built some node (B.I.J). Then there exists some goal (A.i) such that AR^*B holds. Hence there exists a tree with A at the root and B on the left edge. We can therefore extend the sentence so that the part starting with the B reduces to A. One more reduction is now possible, using the rule that gave rise to the goal in the first place. We continue up the tree in this fashion, progressively extending the sentence and satisfying more goals, until we satisfy the Sentence goal. At this point we have the desired sentence. This completes the proof of the claim that every node built has a chance of being used in the fina.1 surface structure.

What does this fancy algorithm buy as far as the practically minded user ts concerned' One thing we do not claim is any improvement over the traditional 0(n) speed limit for parsing sentences of length n. (See [Valiant 1974] for an improvement to this situation - he offers $0(n^{2\,81})$, though practical considerations make it much worse than most 0(n) algorithms with respect to both speed and ability to take advice when parsing "typical" English sentences.) However, we do claim what amounts to an even better improvement in practice than going from $O(n^3)$ to $O(n^2)$, namely an improvement proportional to the number of non-terminals in the grammar. That is, there exist grammars for which the Earley algorithm may generate large state sets in Its operation when our algorithm builds very much fewer nodes. (However, there do not exist any grammars where Ear ley's algorithm runs in time $O(n^{\alpha})$ while ours runs in time $O(n^{\beta})$ for $\beta < \alpha$. There do exist such grammars when comparing our algorithm with the Cocke-Kasami-Younger algorithm)

In even more practical terms, how does this affect parsers working with a purported grammar of English? We conducted an experiment to compare our algorithm with the Cocke-Kasami-Younger algorithm by the simple expedient of suppressing the test in the procedure "note" that makes our algorithm different from the other. Working with the grammar of English used in the "9-hour" French translator exhibited in [Pratt 1973). we found an improvement of a factor of five in the number of nodes built altogether! In fact, with the new algorithm almost all of the nodes built were used in the final surface structures of the sentences we tried. Lack of a local implementation of Earley's algorithm has prevented us from comparing it with ours in an actual machine simulation. However, it does nor require a machine simulation to see that the sort of thing that makes our parser better than Ear ley's in some grammars is exactly what arises in English grammars. For example, if one has the rules Sentence -> Np Vp, Sentence -> Wh Vp and Sentence -> Vp, then Earley's algorithm will generate states corresponding to each of these rules even when the sentence begins with, say, "Why." Earleys algorithm is not smart enough to realize that the first and third rules can be ruled out here (we are making some obvious assumptions about what the rest of this simple grammar might look like).

2.3 ..Taking Advice

One attractive feature of the above technique is that we did not need to "compile" the grammar; we retained the interpretive nature of the Earley and Cocke algorithms. This makes it simple for the user to contribute to the operation of the parser, since all the parser is doing at each step is recognizing that some combination of phrases forms a new phrase The user is given the opportunity at each step to look at the constituents of those phrases, to consult his model of the world, or to perform deductions His conclusions are summarized numerically for LINGOL's benefit, and in pursuing any particular structure, LINGOL accumulates these numbers as a measure of its confidence in that structure. These confidence numbers are used to choose between alternative ambiguous structures. The winning structures are made readily available to the generative component while the losing structures are kept around (on the end of a list of alternatives) in case the generative component becomes dissatisfied with the choice made by the cognitive component and wants to try some of the others.

The style of programming used in the cognitive component is analogous to that described in [Pratt 1973] for the generative component The primary difference is that, since the structures are being built bottom-up at the time the cognitive component is being built, it is not possible to declare variables high up in the tree for use by routines lower down, a facility that gives the generative component considerable power This inability is inherent in the nature of any system that wants to do criticism on the spot without waiting for the rest of the sentence If you don't know what's coming, you can't (other than by guessing) make assumptions about what the higher nodes using the one in question will look like (This is not altogether true - if the relation R* were represented explicitly as a collection of paths in R, it might be possible to set up variables on high as the goals are being generated, provided nodes being discovered below set their own sights on only one goal It is likely that this would add substantially to the overhead of the system, however)

2.4—Using, the.algorith.m_as a contcal device

The emphasis of this section is on nsight rather than mechanics The program paradigm discussed in [Pratt 1973, p 377) supplies the mechanical details of how modules are assembled and how they communicate. Briefly, the surface structure chosen by the algorithm is taken to be the skeleton for a program whose substantive components are LISP functions (the generative component). These functions are associated with the nodes of the tree The tree is then itself taken to be a large expression, and is evaluated. Two types of communication are provided for: functions may "return" values, which are received by their immediate superiors in the tree, and variables (declared local to some subtree) may be used as "mailboxes" for communication up, down or sideways within that subu.ee (reflecting an apparent locality in many linguistic phenomena) A more detailed account of these mechanics is in [Pratt, 1973]

We have here a somewhat unusual programming environment The user is told that he may write arbitrary LISP code for the function at each vertex; provided the original sentence can be reconstructed from the surface structure information, he is no worse off in principle than if he started from scratch However, this easily made point is not the real issue Rather, the user is supposed to assume that the surface structure the parser found is what he thought it would be The insight is that, although a considerable amount of Enguistic processing of the sentence may still remain to be done after the structure is found, that processing will be of the form "what to reply when you see a ..." rather than "where to search for a." That is, there is no longer the emphasis on control (backup techniques, depth-first vs breadrh-first issues, passing environments around, and so on) that characterizes many papers on parsing Instead, the user has to decide, for each of many local situations, what the answer is

It should be obvious from the above that we are advocating:

(i) uniform control of search, using some linguistic information (the context-free component plus the cognitive component) plus a smart parsing algorithm;

(u) non-uniform modular treatment of the remainder of the user's linguistic information

The second of these reflects the structure of a typical grammar written in English for consumption by humans (at least those grammars written before linguistics became confused with mathematics). The phenomena are treated one at a time in the grammar, and the notions of procedure, control, for-loops, recursion, searching and so on never appear Everything is very modular, even isolated much of the time (By "isolated" we mean that the phenomenon does not depend on some other phenomenon for its full explanation, "modular" only refers to the degree of organization and does not exclude inter-module communication.)

It is our hope that this modularity is what makes it easy for grammarians to write large grammar books for human consumption, rather than that the grammars are written In English If so, it may make more efficient the process of telling English to computers, which has been proceeding slowly to date

All this assumes, of course, that we have a reliable structure finder We snnd by our claim in [Pratt 1973. p. 376] that considerably less linguistic information is needed to get the surface structure than for subsequent processing.

2.5 Role of the Context-free component.

Since the trend these days is away from explicit context-free grammars and towards encoding all syntactic information in other ways, it is reasonable to ask why have a separate context-free component The issue is one of efficiency, among other things It has yet to be demonstrated that English is easy to parse As far as we know, a program with a lot of expertise about English Is going to discover a lot of things to say abou' a sentence, most of them presumably being of the form "it can't be this interpretation because. * There is much wishful thinking these days [Marcus 1975. Riesbeck 1975] about being able to ignore entirely the sorts of parses discovered by the Harvard Predictive Analyzer [Kuno 1965] for reasonable English sentences To this author's knowledge, no such wishful thinking has been realized as a program having the linguistic competence of, say, Sager's system [Sager J973). or for that matter the Predictive Analyzer The problem may be that one cannot dismiss these obscure parses on trivial grounds without also eliminating perfectly good parses of other sentences where the corresponding structure is not so peculiar. Unfortunately, there is no evidence to support this one way or the other, and the author wishes to sit on the fence for the time being as far as whether the above wishful thinking can be put into practice (He would like to put it into practice himself, but along with the rest of the world has no idea how.)

Given that one's program can be expected to encounter many competing interpretations of a sentence, and that in many cases it will have to pass non-trivial judgment on rhese cases, it can be very difficult to write a program to deal with much of English LINGOL allows the user to organize his program so that the burden of the book-keeping associated with discovering and comparing all these possibilities is shifted to the system, allowing the user to concentrate on writing code to criticize individual situations M Marcus has suggested to the author that in so doing he is allowing the user to concentrate on the competence aspects of English by supplying him with packaged performance The context-free rules and the critics encode the sort of information one finds in a grammar book, which is competence, while the system knows about good parsing strategies, which is performance

A long range goal in this regard is to develop a high level language version of LINGOL such that grammars may be given to either computers or people Then if the computer "understood" English on the bas's of that grammar, and if people could read it painlessly. It would make an ideal theory of English Since people read procedures painfully slowly if at all. the high level language will have to be considerably less procedural than at present

In our approach, the CF rules function as a crude approximation to English that permits LINGOL to rapidly select from a huge set of possible structures for the sentence a smalt plausible set for more detailed (and expensive) criticism by the cognitive component. The context-free representation for the "crude approximation" is chosen partly because it is not difficult to construct quite good approximations using context-free grammars, and partly because there exist remarkably efficient algorithms for exploring the space of possible surface structures for sentences of context-free grammars, yet that can accept advice on a step by-step basis

This situation of having the system do one's book-keeping was what obtained in the hey-day of context-free parsers, of course One side-effect of the later disenchantment with and abandonment of context-free grammars was to throw the baby out with the bath-water by reverting to doing much or all of the book-keeping oneself Not only does this require an indefinitely larger programming effort, it a ho requires of the programmer considerable sophistication in parsing techniques if his code is to operate as efficiently as one of the better context-free parsing algorithms, especially when that algorithm can cooperate effectively with the user's code in assisting it to reduce the search space even further.

A comparison of our system with that of Woods [1969] is inevitable Where Woods has augmented transition networks, we have augmented context-free grammars. Since basic (i.e. unaugmented) TN's are exactly equivalent to context-free grammars in strong generative capacity, there should in principle be no difference In practice, there are a number of differences One difference is in our parsing algorithm, a property of the implementation rather than of the LINGOL language, which assumes a larger responsibility for determining the flow of control than does Woods'. Another difference is in the language - we take a static view of English inasmuch as we use the seemingly declarative CF notation, whereas Woods uses the seemingly procedural transition networks The procedural flavor becomes substantive when the augments are introduced. Our augments are intended to be mere grammatical critics; Woods' have considerably more to say about the flow of control. As argued earlier, we feel that the static view is more conducive to efficient writing of grammars, so long as the system can take over the efficiency considerations One notable difference is that the LINGOL language has far fewer primitive concepts than does Woods', without losing any of the features of Woods' system The idea here is that the constructs provided explicitly in Woods system have LISP analogues, so why duplicate them? Another difference is our separation of the cognitive and generative components, which we feel is a plus since then the target language issues can be cleanly separated from the cognitive issues This separation does not preclude the sort of interaction with world knowledge advocated in [Winograd 1971]

3. SYNTAX AND EFFICIENCY:

LINGOL has had a chronic identity crisis over the issue of whether it should predominantly rely on syntax in its initial phase This brief section addresses the issue of whether syntax is a necessary part of a computational linguistics program We have in mind here R. Schank's claim that "syntax is not needed to do parsing" For some variety in the usual replies to this sort of claim, we propose that even if Schank is right (thts assumption is local to this section) syntax may be of value in improving the efficiency of the parsing process This point can be easily overlooked both in informal introspection about whether one felt one needed any syntax to parse the sentence and in formal experiments, eg. with the tachistoscope, designed to show that scrambled sentences shown briefly are recalled in their unscrambled form. What is being overlooked here is how long it took to unscramble the sentence

Given grammars language's sentence's initial segment assumed given in order to see no less overlooked than necessary (Translation: To see that no less is overlooked than necessary, assume we are given an initial segment of a sentence of the language of the given grammar.) If you stumbled over this sentence then perhaps it is because the syntax is not there to speed things up for you. Conventional groupings of words have been rearranged in relatively unfamiliar, though not entirely ungrammatical or meaningless, ways and some "noise" words have gone. Nevertheless, with a tittle extra effort you should at least be able to parse the sentence correctly, and after a few passes you will begin to wonder why you ever had any trouble with it at all. Moreover, there seems no obvious reason why a program that could handle the original sentence could not equally welt handle the above version. (I had difficulty restraining myself from replacing it with a more difficult sentence by the time I had typed it up.) The claim is that in the original version, part of the reason why you had less trouble with it was that it was phrased in a very conventional style that *you* have *encountered* frequently, allowing you to go straight to the places where you expect to find the information in the sentence There are "noise" words all along the way, but to see that they are not really all that noisy, try replacing them with other noise words; the effect wilt be somewhat like switching all the street signs when navigating in one's car.

Thus white it is conceivable that one can get by without syntax (though what this means exactly is surely open to debate), even if one does so one is faced with culling out the structure desired from a huge choice without the benefit of syntactic information to reduce the search space.

BIBLIOGRAPHY

1
Aho, A. V. and J. Ullman 1972. The Theory of Parsing, Translation and Compiling, Vol I, Prentice-Hall Inc, New Jersey.

2
Earley, J. 1968. An Efficient Context-free Parsing Algorithm. Ph.D. Thesis, Computer Science Dept, Carnegie-Mellon University, Pittsburgh, Pa.

3
Fahlman, S. 1973. A Planning System for Robot Construction Tasks. AI TR-283, MIT, Cambridge, Massachusetts.

4
Griffiths and Petrick, S. 1965. On the Relative Efficiencies of Context-free Grammar Recognizers, CACM 8, 5, 289.

5
Kuno, S. 1965. The Predictive Analyzer and a Path Elimination Technique. CACM 8, 7, 453-462.

6
Marcus, M. 1975. Diagnosis as a Notion of Grammar. Preprint of a workshop on Theoretical Issues in Natural Language Processing, Cambridge, Mass., 6-10.

7
Pratt, V. R. 1969. Translation of English into Logical Expressions. M.Sc. Thesis, University of Sydney.

8
Pratt, V. R. 1973. A Linguistics Oriented Programming Language. Proceedings of the Third International Joint Conference on Artificial Intelligence, Stanford University, Calif., August 1973.

9
Riesbeck, C. 1975. Computational Understanding. Preprint of a workshop on Theoretical Issues in Natural Language Processing, Cambridge, Mass., 11-16.

10
Sager, N. 1973. The String Parser for Scientific Literature. In Natural Language Processing, Rustin R. (ed), Algorithmics Press, New York.

11
Schank, R., L. Tesler and S. Weber. 1970. Spinoza II - Conceptual Case Based Natural Language Analysis. AI memo 109, Stanford University, Stanford, California.

12
Thorne, J., P. Bratley and H. Dewar. 1968. The Syntactic Analysis of English by Machine. In Machine Intelligence 3, Michie D. (ed).

13
Valiant, L. 1974. Technical Report on a fast context-free parsing algorithm, Carnegie-Mellon University, Pittsburgh, Pa.

14
Winograd, T. 1971. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Project MAC TR-84, MIT, Cambridge, Massachusetts.

15
Woods, W. A. 1969. Augmented Transition Networks for Natural Language Analysis. Report No CS-1 to the NSF, Aiken Computation Laboratory, Harvard University, Cambridge, Massachusetts.